Method of simulating a circuit which is described in EDIF using a VHDL simulator on a computer

Patent Number: DE4408106
Publication date: 1994-12-15

Inventor(s): LANGE EBERHARD DR
Applicant(s): SIEMENS AG (DE)

Requested Patent: ☐ DE4408106
Application DE19944408106 19940310
Priority Number(s): EP19930109203 19930608
IPC Classification: G06F15/60
EC Classification: G06F17/50C3
Equivalents:

Abstract

With the invention, a method of simulating EDIF circuits on a VHDL simulator is described. By applying the method, e.g. with PLDs, development time can be saved, since the simulation models are available earlier than before. Further advantages are the ability to simulate several circuits together and the simplified search for errors.

DOCKET NO: GR 99 5005
SERIAL NO: 09/680,370
APPLICANT: Schneider

LERNER AND GREENBERG P.A.
P.O. BOX 2480
HOLLYWOOD, FLORIDA 33022
TEL. (954) 925-1100

(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENTAMT

(12) **Offenlegungsschrift**

(10) **DE 44 08 106 A 1**

(51) Int. Cl.[5]: G 06 F 15/60

(21) Aktenzeichen: P 44 08 106.5
(22) Anmeldetag: 10. 3. 94
(43) Offenlegungstag: 15. 12. 94

DE 44 08 106 A 1

(30) Unionspriorität: (32) (33) (31)
08.06.93 EP 93 10 9203.5

(71) Anmelder:
Siemens AG, 80333 München, DE

(72) Erfinder:
Lange, Eberhard, Dr., 90587 Obermichelbach, DE

(54) Verfahren zur Simulation einer in EDIF beschriebenen Schaltung mit einem VHDL-Simulator auf einem Rechner

(57) Mit der Erfindung wird ein Verfahren zur Simulation von EDIF-Schaltungen auf einem VHDL-Simulator beschrieben. Durch Anwendung des Verfahrens, beispielsweise bei PLDs kann man Entwicklungszeit sparen, da man die Simulationsmodelle früher als bisher zur Verfügung hat. Ein weiterer Vorteil ist die Simulierbarkeit von mehreren Schaltungen gemeinsam und die erleichterte Fehlersuche.

DE 44 08 106 A 1

Beschreibung

Die Simulation elektronischer Baugruppen auf EDV-Anlagen hat sich in den letzten Jahren zu einer wichtigen Entwicklungstechnik entwickelt. Der übliche Entwicklungsweg, das nachträgliche Testen einer Schaltung, wird durch die fortschreitende Miniaturisierung bei innovativen Bauteilen zunehmend schwieriger. Der Weg über die Simulation kann hier zumindest zum Teil Abhilfe schaffen. Es wird auch immer wieder festgestellt, daß der Einsatz der Simulation die Entwicklungszeit der Schaltungen verkürzt, die Qualität verbessert und somit insgesamt zu einer Steigerung der Produktivität führt. Der Verbreitungsgrad der Simulation ist dennoch insgesamt gering. Das zentrale Hemmnis technischer Natur ist das Fehlen von geeigneten Simulationsmodellen für die entsprechenden Bauteile. Dies ist insbesondere bei innovativen Bauteilen wie zum Beispiel programmierbaren Logikbausteinen (PLDs) der Fall.

Die Güte eines Simulationsmodells wird unter anderem durch seine Leistungsfähigkeit und die Möglichkeiten der Lokalisierung von Fehlverhalten bestimmt.

Bei der Beschreibung von Bau- oder Schaltungsteilen durch Netzlisten ist der EDIF (Electronic Design Interchange Format)-Standard gebräuchlich. Für viele Bauteile liegt daher eine Netzlistenbeschreibung in Form von EDIF vor. VHDL (VHSIC Hardware Description Language) ist eine Verhaltensbeschreibungssprache, die sowohl bei der Entwicklung von ASICs als auch bei der Beschreibung von Bauteilen für Simulationsmodelle einer der führenden Standards ist. Moderne Simulatoren für digitale Baugruppen sind in der Lage, VHDL-Modelle zu simulieren.

Diese beiden Standards sind in [1] und [2] festgelegt.

Bei fehlenden Modellen konnte bisher überhaupt nicht oder nur in reduzierter Form (d. h. ohne das innovative Bauteil) simuliert werden. Dies hat zur Folge, daß durch Redesigns eine Verlängerung der Entwicklungszeit oder Qualitätseinbußen entstehen.

Die der Erfindung zugrundeliegende Aufgabe besteht darin, ein Verfahren anzugeben, mit dem in EDIF beschriebene Schaltungen auf einem VHDL-Simulator simuliert werden können.

Diese Aufgabe wird gemäß den Merkmalen des Anspruches 1 gelöst.

Weiterbildungen der Erfindung ergeben sich aus den Unteransprüchen.

Es wurde ein Arbeitsverfahren entwickelt, mit dem EDIF-Netzlisten (EDIF Version 2 0 0) in VHDL-Modelle (IEEE Std 1076—1987) übertragen werden.

Durch den Einsatz des erfindungsgemäßen Verfahrens erzielt man verschiedene Vorteile:
Zum einen sind die Schaltungen effizienter simulierbar, da VHDL-Simulatoren umfangreiche Simulationsmöglichkeiten bieten. Aus demselben Grund gestaltet sich die Fehlersuche innerhalb der Schaltung wesentlich einfacher.
Zum anderen beschleunigt die Zuordnung einer EDIF-Instanz zu einem VHDL-Prozeß den Simulationsvorgang. VHDL-Prozesse können vom Simulator parallel abgearbeitet werden. In Verbindung mit einer prozeßspezifischen Sensitivity-Liste bedeutet dies, daß nur solche Prozesse im Simulator berechnet werden, deren Eingangssignale sich ändern.

Die Schaltung wird so also parallelisiert und kann damit ohne Qualitätsverlust wesentlich schneller simuliert werden.

EDIF-Netzlisten bestehen im wesentlichen aus Instanzen (EDIF-Schlüsselwort "Instance"), die zum Beispiel logische Gatter oder Flip-Flops repräsentieren, und Netzen (EDIF-Schlüsselwort "Net"), welche die Verbindungen zwischen den Instanzen darstellen. Solche Instanzen können beispielsweise logische Schaltglieder, wie AND, OR, NOT, XOR, Tristate-Puffer, oder Latch-Schaltglieder sein. Die Anzahl der Eingänge ist dabei beliebig. Weiterhin können beispielsweise getaktete oder getriggerte D- bzw. JK-Flip-Flop's verwendet werden. Zusätzliche Möglichkeiten bei der Beschreibung von Schaltungen ergeben sich durch den Einsatz von beispielsweise Filter- oder Verzögerungs-Elementen, zur Definition des Zeitverhaltens einer Schaltung. Das zeitliche und logische Verhalten der Instanzen wird beispielsweise mit Hilfe einer externen oder internen Bibliothek von Grundfunktionen beschrieben. Die Schnittstelle einer Instanz (Ein-/Ausgänge) wird in der EDIF-Netzliste definiert.

Der Kern des Verfahrens beschäftigt sich mit einer schematischen und effektiven Umsetzung der Schnittstellen der Instanzen und der Netze in VHDL-Modelle:

- Jeweils eine EDIF-Instanz wird in einen VHDL-Prozeß umgesetzt (1).
- Jedem EDIF-Netz wird ein VHDL-Signal zugeordnet (2).

(1) Da die Namen der Instanzen in der EDIF-Netzliste eindeutig sind, können sie direkt als Name des VHDL-Prozesses verwendet werden. Die einer Instanz zugeordnete Grundfunktion (UND-/ODER-Gatter etc.) kann aus der EDIF-Beschreibung entnommen werden und läßt sich mit VHDL-Sprachelementen modellieren. Das logische Verhalten der Grundfunktion muß dabei aus der Bibliothek von Grundfunktionen entnommen werden.

(2) Für die Eingänge und Ausgänge einer Instanz sind in der EDIF-Netzliste nicht notwendigerweise global eindeutige Namen vorhanden. Lediglich innerhalb einer Instanz ist die Benamung eindeutig (z. B. Numerierung). Die im VHDL-Modell benötigten eindeutigen Namen für Signale können wie folgt durchs Zusammensetzen erzeugt werden: Ein EDIF-Netz verbindet jeweils den Ausgang einer Instanz (Start-Instanz) mit einem oder mehreren Eingängen eines oder mehrerer Instanzen (Ziel-Instanz). Das dem EDIF-Netz zugeordnete VHDL-Signal erhält den durch Zusammensetzen des Namens der Start-Instanz mit dem Namen des Ausgangs entstehenden Namen.

Durch diese Art der Benamung der Netze können auch Verbindungen zwischen einem Ausgang einer

Start-Instanz und mehreren Eingängen von Ziel-Instanzen im VHDL-Modell korrekt implementiert werden. Bei der Benamung der Prozesse und Signale muß darauf geachtet werden, daß in VHDL verbotene Zeichen (z. B.: "&") durch andere Zeichen ersetzt werden.

Ein VHDL-Prozeß erhält als "Sensitivity List" nur die Signale, die tatsächlich Eingänge sind. Dies hat zur Folge, daß der Prozeß nur dann aktiv wird, wenn eine Änderung der Eingangssignale stattfindet. Da auf diese Weise nur die Prozesse aktiviert werden, die tatsächlich momentan benötigt werden, ist eine gute Leistungsfähigkeit der Modelle gewährleistet.

VHDL-Modelle sind von der Struktur der Sprache her relativ leicht lesbar. Zudem stehen bei den meisten Simulatoren leistungsfähige Debugger zur Verfügung. Dadurch können auftretende Entwicklungsfehler mit Hilfe der VHDL-Modelle sehr leicht lokalisiert und bis auf Gatterebene zurückverfolgt werden.

Das Verfahren wurde am Beispiel käuflich erwerbbarer programmierbarer Logikbausteine erfolgreich erprobt. Dafür wurde ein DV-Programm in der Programmiersprache C implementiert, was eine Konvertierung der EDIF-Netzlisten in VHDL-Modelle nach dem oben beschriebenen Übertragungsverfahren durchführt. Die VHDL-Modelle wurden mit Hilfe einer kommerziell vertriebenen Simulationsumgebung getestet. Eine simultane Simulation von bis zu 8 PLDs konnte mit guter Performanz durchgeführt werden.

Im folgenden wird die Erfindung anhand einer Figur weiter erläutert.

Die **Fig. 1** zeigt eine Schaltung von drei EDIF-Instanzen, welche simuliert werden sollen. Mit and1_3 ist ein Und-Gatter und mit latch_4 ein Latch bezeichnet. Beide sind über Signalleitungen N_8 und N_9 mit einer Tri-State-Instanz tri_2 verbunden. In den weiter hinten folgenden Beschreibungsteilen "EDIF-Netzliste" und "VHDL-Modell" ist diese Schaltung im jeweiligen Format beschrieben.

Die EDIF-Netzliste wurde mit einem käuflichen EPLD-Entwicklungstool erzeugt. Das VHDL-Modell kann direkt mit dem Digitalsimulator simuliert werden.

1. Beschreibung der EDIF-Netzliste

Hier wird zunächst die "EDIF-Netzliste" beschrieben. Die zugehörige Beschreibung des VHDL-Modells findet sich unter analoger Numerierung in der "Beschreibung des VHDL-Modells".

1.1 Bibliotheks-Vereinbarungen

In der EDIF-Netzliste werden zunächst einige für die Umsetzung in ein VHDL-Modell nicht relevante Vereinbarungen getroffen:
edifVersion, edifLevel, KeywordMap, status.

Mit der Vereinbarung "library" wird festgelegt, welche Primitives aus der Bibliothek benötigt werden. Aufgeführt werden außerdem das jeweilige Interface und teilweise auch Verzögerungs- bzw. Setup- und Hold-Zeiten. Zunächst wird mit

scale 1 (e 1 -10) (unit TIME)

das Zeitraster für alle nachfolgenden Zeitangaben auf 100 ps definiert. Die Signale können die logischen Werte L, H, X, Z annehmen.

Mit der Vereinbarung

cell AND1

wird festgelegt, daß für das EPLD ein UND-Gatter mit einem Eingang benötigt wird. Das Interface besteht aus einem Eingang ("INPUT") und einem Ausgang ("OUTPUT"), die über

```
( port  &2
                ( direction INPUT )  )
( port &1
                ( direction OUTPUT ) )
```

definiert werden. Analoges gilt für die Vereinbarungen "cell DELAY", "cell OR1" und "cell XOR2". Mit der Vereinbarung

cell LATCH

wird ein Latch definiert. Sein Interface besteht gemäß

```
( port D
     ( direction INPUT ) )
( port ENA
     ( direction INPUT ) )
( port Q
     ( direction OUTPUT ) )
```

aus den Eingängen "Enable", "Daten" sowie dem Ausgang "Q". Da für das Latch die Verzögerungs- bzw. Setup- und Hold-Zeiten global für das gesamte EPLD einheitlich sind, werden sie ebenfalls hier festgelegt. Mit der Vereinbarung

```
( pathDelay
     ( delay 40 )
     ( event
          ( portRef ENA )
          ( transition L H ) )
     ( event
          ( portRef Q )
          ( transition
               ( logicOneOf L H )
               ( logicOneOf L H ) ) ) )
```

wird die Verzögerungszeit von einer steigenden Flanke (Übergang von Low zu High) auf dem Enable Eingang bis zur Änderung am Ausgang (von Low zu High oder von High zu Low) auf 40 × 100 ps = 4 ns festgelegt. Analog werden über

```
( forbiddenEvent
     ( timeInterval
          ( offsetEvent
               ( event
                    ( portRef ENA )
                    ( transition L H ) )
               - 100 )
          ( duration 100 ) )
     ( event
          ( portRef D ) ) )
```

die Setup-Zeit (Daten bezüglich Enable) zu 10 ns und über

```
( forbiddenEvent
    ( timeInterval
        ( event
            ( portRef ENA )
            ( transition L H ) )
        ( duration 100 ) )
    ( event
        ( portRef D ) ) )
```

die Hold-Zeit (Daten bezüglich Enable) zu 10 ns festgelegt.
Mit der Vereinbarung "cell TRI" wird schließlich ein Tristate-Treiber mit der Verzögerungszeit 13 ns definiert.

1.2 Interface-Vereinbarungen

Mit "cell TEST" beginnt nun die eigentliche Beschreibung des EPLD-Designs mit dem Namen "TEST".
Unter der Vereinbarung "interface" werden neben dem Typ des EPLD

(designator "EPM5128 J 68")

die globalen Ein- und Ausgänge des EPLD festgelegt:

```
( port VCC
    ( direction INPUT ) )
( port GND
    ( direction INPUT ) )
( port ( rename P_66 "ENA" )
    ( direction INPUT )
    ( designator "66"" ) )
( port ( rename P_68 "D" )
    ( direction INPUT )
    ( designator "68" ) )
( port ( rename P_65 "OUT" )
    ( direction OUTPUT )
    ( designator "65" ) )
```

Dabei bezeichnen P_66, P_68 und P_65 die internen Signalnamen, die eine Zuordnung zu den entsprechenden Netzen möglich machen (siehe unten). Der jeweilige "designator" bezeichnet die Nummer des physikalischen Pins.

1.3 Instance-Vereinbarungen

Hier wird festgelegt, aus welchen Gattern, Latches und Treibern das Design des EPLD tatsächlich besteht ("contents"). Mit

```
( instance tri_2
    (viewRef view1
        ( cellRef TRI ) ) )
```

wird ein "instance" mit dem eindeutigen Namen tri_2 vom Typ TRI (also ein Tristate-Treiber, siehe 2.) definiert. Bei einigen "instance" werden zusätzlich noch die Verzögerungszeiten festgelegt:

```
( instance and1_3
    ( viewRef view1
        ( cellRef AND1 ) )
    (portInstance &1
        ( port Delay
            ( derivation CALCULATED )
            ( delay 130 ) ) ) )
```

Damit wird ein UND-Gatter mit dem Namen and1_3 bezeichnet und eine Verzögerungszeit von 13 ns zugeordnet.

1.4 Net-Vereinbarungen

Die "net"-Vereinbarung definiert Verbindungen zwischen den Ein- und Ausgängen ("ports") der zuvor festgelegten "instance".

```
( net ( rename N_1 "VCC" )
    ( joined
        ( portRef VCC )
        ( portRef &2
            ( instanceRef and1_3 ) ) ) )
```

Das Netz mit dem Namen N_1 verbindet also den globalen Ausgang VCC mit dem "port" &2 des UND-Gatters mit dem Namen and1_3. Ein weiteres Beispiel

```
( net N_8
    ( joined
        ( portRef &1
            ( instanceRef and1_3 ) )
        ( portRef OE
            ( instanceRef tri_2 ) ) ) )
```

verbindet den Ausgang &1 des UND-Gatters mit dem Namen and1_3 mit dem Output Enable Eingang (OE) des Tristate-Treibers mit dem Namen tri_2.

2. Beschreibung des VHDL-Modells

Hier folgt die Beschreibung des aus der umgesetzten EDIF-Netzliste entstandenen VHDL-Modells. Da die VHDL-Prozesse dabei sowohl Information aus Instance- als auch aus Netz-Vereinbarungen enthalten, werden diese zusammengefaßt in Kapitel 2.3/2.4 beschrieben.

2.1 Bibliotheks-Vereinbarungen

Auch im VHDL-Modell werden mit "library" bzw. "use" einige Bibliotheksvereinbarungen getroffen. Einige häufig benutzte Funktionen (z. B. Fehlermeldung, Test aufsteigende Flanke) werden über "procedure" bzw. "function" definiert.

2.2 Interface-Vereinbarungen

Im VHDL-Modell werden die globalen Ein- und Ausgänge über die "entity" festgelegt:

```
entity test is
port(
     VCC : in qsim_state ;
     GND : in qsim_state ;
     ENA : in qsim_state ;
     D : in qsim_state ;
     OUT : out qsim_state) ;
end test ;
```

Dabei ist "qsim_state" ein Signal, welches der Quicksim II versteht, und das die logischen Werte X, Z, 1, 0 annehmen kann.

2.3/2.4 Instance-Vereinbarungen/Net-Vereinbarungen

Im VHDL-Modell werden alle intern benötigten Signalnamen deklariert. Ein internes Signal entspricht jeweils einem Netz und erhält den Namen des "instance" zusammengesetzt mit dem Namen des Ausganges des "instance". Das Zeichen "&" darf in VHDL nicht benutzt werden und wird daher durch ein "P" ersetzt. Damit wird das "net" N_8 durch das Signal

signal and1_3_P1: qsim_state;

wiedergegeben.

Das "net" N_1 stellt die Verbindung zu einem globalen Eingang (VCC) her und muß daher nicht zusätzlich intern deklariert werden.

Im VHDL-Modell wird jedes "instance" aus der EDIF-Netzliste durch einen "process" dargestellt. Das UND-Gatter mit dem Namen and1_3 wird durch einen "process" mit dem gleichen Namen realisiert:

```
and1_3 : process(VCC)
begin
     and1_3_P1 <= transport VCC after 13000ps ;
end process and1_3 ;
```

Der Prozeß wird nur aktiv, wenn sich das Eingangssignal VCC ändert. Das Signal and1_3_P1 erhält dann den Wert des Signals VCC und dient gleichzeitig als Eingang für den Tristate-Treiber mit dem Namen tri_2:

tri_2: process(latch_4_Q, and1_3_P1).

In diesem Tristate-Treiber erhält der globale Ausgang OUT seinen Wert:

OUT < = transport latch_4_Q after 6000 ps;

Es folgt die "EDIF-Netzliste"

```
( edif TEST
        ( edifVersion 2 0 0 )
        ( edifLevel 0 )
        ( keywordMap
                ( keywordLevel 0 ) )
        ( status
                ( written
                        ( timeStamp 1992 11 5 1 0 42 )
                        ( program "MAX+PLUS II VERSION 2.11  04/06/92"
                                ( version "ALTERA EDIF 3.0" ) ) ) )
        ( library ALTERA
                ( edifLevel 0 )
                ( technology
                        ( numberDefinition
                                ( scale 1 ( e 1 -10 ) ( unit TIME ) ) )
                        ( simulationInfo
                                ( logicValue L )
                                ( logicValue H )
                                ( logicValue X )
                                ( logicValue Z ) ) )
                ( cell AND1
                        ( cellType GENERIC )
```

```
        (view view1
                (viewType NETLIST)
                (interface
                        (port &2
                                (direction INPUT))
                        (port &1
                                (direction OUTPUT)))))
(cell DELAY
        (cellType GENERIC)
        (view view1
                (viewType NETLIST)
                (interface
                        (port &2
                                (direction INPUT))
                        (port &1
                                (direction OUTPUT)))))
(cell LATCH
        (cellType GENERIC)
        (view view1
                (viewType NETLIST)
                (interface
                        (port D
                                (direction INPUT))
                        (port ENA
                                (direction INPUT))
                        (port Q
                                (direction OUTPUT))
                        (timing
                                (derivation CALCULATED)
                                (pathDelay
                                        (delay 40)
                                        (event
                                                (portRef ENA)
                                                (transition L H))
                                        (event
                                                (portRef Q)
                                                (transition
                                                        (logicOneOf L H)
                                                        (logicOneOf L H))))
                                (forbiddenEvent
                                        (timeInterval
                                                (offsetEvent
                                                        (event
                                                                (portRef ENA)
```

```
                                                    (transition L H))
                                                -100)
                                            (duration 100))
                                    (event
                                            (portRef D)))
                                (forbiddenEvent
                                    (timeInterval
                                            (event
                                                    (portRef ENA)
                                                    (transition L H))
                                            (duration 100))
                                    (event
                                            (portRef D)))))))
(cell OR1
        (cellType GENERIC)
        (view view1
                (viewType NETLIST)
                (interface
                        (port &2
                                (direction INPUT))
                        (port &1
                                (direction OUTPUT)))))
(cell TRI
        (cellType GENERIC)
        (view view1
                (viewType NETLIST)
                (interface
                        (port IN
                                (direction INPUT))
                        (port OE
                                (direction INPUT))
                        (port OUT
                                (direction OUTPUT)
                                (portDelay
                                        (derivation CALCULATED)
                                        (delay 60)))
                        (timing
                                (derivation CALCULATED)
                                (pathDelay
                                        (delay 130)
                                        (event
                                                (portRef OE)
                                                (transition H L))
                                        (event
```

```
                                    (portRef OUT)
                                    (transition
                                        (logicOneOf L H) Z)))
                            (pathDelay
                                (delay 130)
                                (event
                                    (portRef OE)
                                    (transition L H))
                                (event
                                    (portRef OUT)
                                    (transition Z
                                        (logicOneOf L H)))))))))
(cell XOR2
    (cellType GENERIC)
    (view view1
        (viewType NETLIST)
        (interface
            (port &2
                (direction INPUT))
            (port &3
                (direction INPUT))
            (port &1
                (direction OUTPUT)))))
(cell TEST
    (cellType GENERIC)
    (view view1
        (viewType NETLIST)
        (interface
            (designator "EPM5128 J 68")
            (port VCC
                (direction INPUT))
            (port GND
                (direction INPUT))
            (port (rename P_66 "ENA")
                (direction INPUT)
                (designator "66"))
            (port (rename P_68 "D")
                (direction INPUT)
                (designator "68"))
            (port (rename P_65 "OUT")
                (direction OUTPUT)
                (designator "65")))
        (contents
            (instance tri_2
```

```
        (viewRef view1
                (cellRef TRI) ))
(instance and1_3
        (viewRef view1
                (cellRef AND1) )
        (portInstance &1
                (portDelay
                        (derivation CALCULATED)
                        (delay 130) ) ) )
(instance latch_4
        (viewRef view1
                (cellRef LATCH) ) )
(instance xor2_5
        (viewRef view1
                (cellRef XOR2) )
        (portInstance &1
                (portDelay
                        (derivation CALCULATED)
                        (delay 30) ) ) )
(instance or1_6
        (viewRef view1
                (cellRef OR1) )
        (portInstance &1
                (portDelay
                        (derivation CALCULATED)
                        (delay 0) ) ) )
(instance and1_7
        (viewRef view1
                (cellRef AND1) )
        (portInstance &1
                (portDelay
                        (derivation CALCULATED)
                        (delay 130) ) ) )
(instance delay_8
        (viewRef view1
                (cellRef DELAY) )
        (portInstance &1
                (portDelay
                        (derivation CALCULATED)
                        (delay 90) ) ) )
(instance and1_9
        (viewRef view1
                (cellRef AND1) )
        (portInstance &1
```

```
                (portDelay
                        (derivation CALCULATED)
                        (delay 130))))
(instance delay_10
        (viewRef view1
                (cellRef DELAY))
        (portInstance &1
                (portDelay
                        (derivation CALCULATED)
                        (delay 50))))
(instance and1_11
        (viewRef view1
                (cellRef AND1))
        (portInstance &1
                (portDelay
                        (derivation CALCULATED)
                        (delay 130))))
(instance delay_12
        (viewRef view1
                (cellRef DELAY))
        (portInstance &1
                (portDelay
                        (derivation CALCULATED)
                        (delay 90))))
(net (rename N_1 "VCC")
        (joined
                (portRef VCC)
                (portRef &2
                        (instanceRef and1_3))))
(net (rename N_2 "GND")
        (joined
                (portRef GND)
                (portRef &2
                        (instanceRef and1_9))))
(net (rename N_7 "OUT")
        (joined
                (portRef P_65)
                (portRef OUT
                        (instanceRef tri_2))))
(net N_8
        (joined
                (portRef &1
                        (instanceRef and1_3))
                (portRef OE
```

```
                    (instanceRef tri_2 ) ) ) )
(net N_9
    (joined
        (portRef Q
            (instanceRef latch_4 ) )
        (portRef IN
            (instanceRef tri_2 ) ) ) )
(net N_10
    (joined
        (portRef &1
            (instanceRef xor2_5 ) )
        (portRef D
            (instanceRef latch_4 ) ) ) )
(net N_11
    (joined
        (portRef &1
            (instanceRef or1_6 ) )
        (portRef &2
            (instanceRef xor2_5 ) ) ) )
(net N_12
    (joined
        (portRef &1
            (instanceRef and1_7 ) )
        (portRef &2
            (instanceRef or1_6 ) ) ) )
(net N_13
    (joined
        (portRef &1
            (instanceRef delay_8 ) )
        (portRef &2
            (instanceRef and1_7 ) ) ) )
(net (rename N_14 "D" )
    (joined
        (portRef P_68 )
        (portRef &2
            (instanceRef delay_8 ) ) ) )
(net N_15
    (joined
        (portRef &1
            (instanceRef and1_9 ) )
        (portRef &3
            (instanceRef xor2_5 ) ) ) )
(net N_16
    (joined
```

```
                    (portRef &1
                        (instanceRef delay_10))
                    (portRef ENA
                        (instanceRef latch_4))))
            (net N_17
                (joined
                    (portRef &1
                        (instanceRef and1_11))
                    (portRef &2
                        (instanceRef delay_10))))
            (net N_18
                (joined
                    (portRef &1
                        (instanceRef delay_12))
                    (portRef &2
                        (instanceRef and1_11))))
            (net (rename N_19 "ENA")
                (joined
                    (portRef P_66)
                    (portRef &2
                        (instanceRef delay_12)))))))))
    (design ROOT
        (cellRef TEST
            (libraryRef ALTERA))))
```

Es folgt das "VHDL-Modell"

```
library mgc_portable;
library std;
use mgc_portable.qsim_logic.all;
use mgc_portable.qsim_relations.all;
use std.textio.all;

entity test is
port(
  VCC : in qsim_state ;
  GND : in qsim_state ;
  ENA : in qsim_state ;
  D : in qsim_state ;
```

```
    OUT : out qsim_state) ;
end test ;

architecture edif of test is

 file error_file: text is out "TEST.log" ;

 procedure write_err_msg(error_message: string(1 to 50)) is
  variable dummy_line : line ;
  begin
   write(dummy_line,error_message) ;
   write(dummy_line,now) ;
   writeline(error_file,dummy_line) ;
  end write_err_msg ;

 procedure write_sht_msg(by_time : IN time ; error_message: string(1 to 50)) is
  variable dummy_line : line ;
  variable by_string  : string(1 to 4) := " by " ;
  begin
   write(dummy_line,error_message) ;
   write(dummy_line,now) ;
   write(dummy_line,by_string) ;
   write(dummy_line,by_time) ;
   writeline(error_file,dummy_line) ;
  end write_sht_msg ;

 function undeftest  (signal test_signal : in qsim_state) return boolean is
  variable undefined : boolean := false ;
  begin
   if (now > 0.0ns) then
    if (not (test_signal = '0' or test_signal = '1')) then
     undefined := true ;
    end if ;
   end if ;
   return (undefined) ;
  end undeftest ;

 function rise(signal test_signal: in qsim_state) return boolean is
  variable rising : boolean := false ;
  begin
   if (test_signal'event and (test_signal = '1') and (test_signal'last_value = '0'))
   then
    rising := true ;
   end if ;
```

```
  return (rising) ;
 end rise ;

function fall(signal test_signal: in qsim_state) return boolean is
 variable falling : boolean := false ;
 begin
  if (test_signal'event and (test_signal = '0') and (test_signal'last_value = '1'))
  then
   falling := true ;
  end if ;
  return (falling) ;
 end fall ;

function to_high(signal test_signal: in qsim_state) return boolean is
 variable rising : boolean := false ;
 begin
  if (test_signal'event and (test_signal = '1') and (not(test_signal'last_value = '1')))
  then
   rising := true ;
  end if ;
  return (rising) ;
 end to_high ;

function to_low(signal test_signal: in qsim_state) return boolean is
 variable falling : boolean := false ;
 begin
  if (test_signal'event and (test_signal = '0') and (not(test_signal'last_value = '0')))
  then
   falling := true ;
  end if ;
  return (falling) ;
 end to_low ;

signal and1_3_P1 : qsim_state ;
signal latch_4_Q : qsim_state ;
signal xor2_5_P1 : qsim_state ;
signal or1_6_P1 : qsim_state ;
signal and1_7_P1 : qsim_state ;
signal delay_8_P1 : qsim_state ;
signal and1_9_P1 : qsim_state ;
signal delay_10_P1 : qsim_state ;
signal and1_11_P1 : qsim_state ;
signal delay_12_P1 : qsim_state ;
```

```
begin

 tri_2 : process(latch_4_Q,and1_3_P1)
 begin

  if (to_low(and1_3_P1)) then
   OUT <= transport 'Z' after 13000 ps ;
  elsif (to_high(and1_3_P1)) then
   OUT <= transport latch_4_Q after 13000 ps ;
  else
   if (and1_3_P1 = '1') then
    if (and1_3_P1'last_event < 13000 ps) then
     OUT <= transport latch_4_Q after (13000 ps - and1_3_P1'last_event) ;
    else
     OUT <= transport latch_4_Q after 6000 ps ;
    end if ;
   end if ;
   if (and1_3_P1 = '0' and 13000 ps - and1_3_P1'last_event > 6000 ps) then
    OUT <= transport latch_4_Q after 6000 ps, 'Z' after (13000 ps - and1_3_P1'last_event) ;
   end if ;
  end if ;
end process tri_2 ;

and1_3 : process(VCC)
begin
 and1_3_P1 <= transport VCC after 13000 ps;
end process and1_3 ;

latch_4 : process(xor2_5_P1,delay_10_P1)
 variable setup_time_violation : string(1 to 39) := "setup time violation at latch_4 @ TIME " ;
 variable hold_time_violation : string(1 to 38) := "hold time violation at latch_4 @ TIME " ;
begin

 if (rise(delay_10_P1)) then
  latch_4_Q <= transport xor2_5_P1 after 4000 ps ;
 elsif (fall(delay_10_P1)) then
  if (xor2_5_P1'last_event < 10000 ps) then
   write_sht_msg(10000 ps -xor2_5_P1'last_event,setup_time_violation) ;
  end if ;
  latch_4_Q <= transport xor2_5_P1'delayed(4000 ps) after 4000 ps;
 elsif ((xor2_5_P1'event) and (delay_10_P1 = '1')) then
  if (delay_10_P1'last_event < 10000 ps) then
   write_sht_msg(10000 ps - delay_10_P1'last_event,hold_time_violation) ;
  end if ;
```

```
  latch_4_Q <= transport xor2_5_P1 after 4000 ps;
 end if ;
 if (undeftest(delay_10_P1) and (xor2_5_P1 /= latch_4_Q)) then
  latch_4_Q <= transport 'X' after 4000 ps ;
 end if ;

end process latch_4 ;

xor2_5 : process(or1_6_P1,and1_9_P1)
begin
 xor2_5_P1 <= transport or1_6_P1 xor and1_9_P1 after 3000 ps;
end process xor2_5 ;

or1_6 : process(and1_7_P1)
begin
 or1_6_P1 <= transport and1_7_P1 after 0 ps;
end process or1_6 ;

and1_7 : process(delay_8_P1)
begin
 and1_7_P1 <= transport delay_8_P1 after 13000 ps;
end process and1_7 ;

delay_8 : process(D)
begin
 delay_8_P1 <= transport D after 9000 ps;
end process delay_8 ;

and1_9 : process(GND)
begin
 and1_9_P1 <= transport GND after 13000 ps;
end process and1_9 ;

delay_10 : process(and1_11_P1)
begin
 delay_10_P1 <= transport and1_11_P1 after 5000 ps;
end process delay_10 ;

and1_11 : process(delay_12_P1)
begin
 and1_11_P1 <= transport delay_12_P1 after 13000 ps;
end process and1_11 ;

delay_12 : process(ENA)

begin
 delay_12_P1 <= transport ENA after 9000 ps;
end process delay_12 ;

end edif ;
```

Bezugszeichenliste

and1_3 UND-Gatter
latch_4 latch-Gatter
tri_2 tristate-Gatter
&1 Ausgang am UND-Gatter
Q Ausgang am latch-Gatter
OE Eingang am tristate Gatter
IN Eingang am tristate Gatter
N_8 Signalleitungsnamen
N_9 Signalleitungsnamen

Literatur

[1] Electronic Design Interchange Format, Version 2 0 0, Recommended Standard EIA-548, ANSI/EIA-548-1988, Electronic Industries Association, Washington D.C., USA, 1990.
[2] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076—1987, New York, USA, 1988.

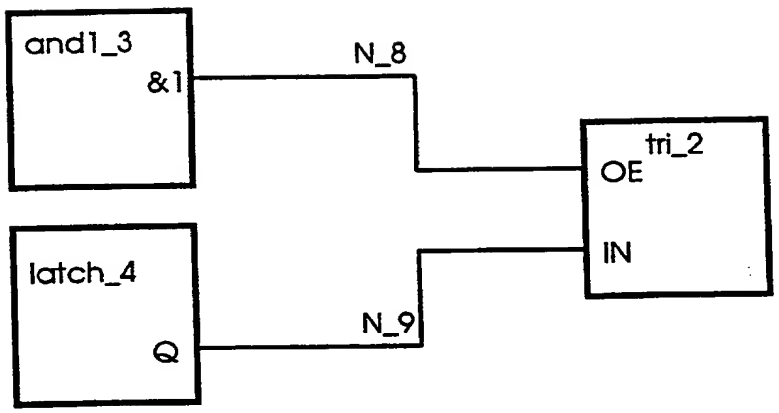
Patentansprüche

1. Verfahren zur Simulation einer in EDIF beschriebenen Schaltung mit einem VHDL-Simulator auf einem Rechner,
   a) bei dem jeweils einer EDIF-Instanz genau ein VHDL-Prozeß mit der gleichen Schaltungsfunktion zugeordnet wird,
   b) und bei dem diese VHDL-Prozesse auf dem VHDL-Simulator simuliert werden.
2. Verfahren nach Anspruch 1, bei dem mindestens ein VHDL-Prozeß unter Verwendung einer Programm-Bibliothek für Schaltungs-Grundelemente generiert wird.
3. Verfahren nach einem der Ansprüche 1 oder 2, bei dem mindestens ein VHDL-Prozeß durch ein unspezifisches C-Programm generiert wird, wobei mindestens die Zahl der Ein- und/oder Ausgänge der Schaltungsfunktion durch die EDIF-Instanz bestimmt wird.
4. Verfahren nach einem der Ansprüche 2 oder 3, bei dem das Zeitverhalten mindestens eines Schaltungs-Grundelementes durch die EDIF-Instanz bestimmt wird.
5. Verfahren nach einem der Ansprüche 1 bis 4, bei dem jedem EDIF-Netz mindestens ein VHDL-Signal zugeordnet wird.
6. Verfahren nach Anspruch 5, bei dem ein VHDL-Signalname aus der Bezeichnung der EDIF-Instanz, von der das zu benennende Signal ausgeht, und der Bezeichnung des Signalausgangs der EDIF-Instanz gebildet wird.
7. Verfahren nach einem der obigen Ansprüche, bei dem der VHDL-Prozeß ebenso wie die zugeordnete EDIF-Instanz benannt wird.
8. Verfahren nach einem der obigen Ansprüche, bei dem in eine Sensitivity-Liste mindestens eines VHDL-Prozesses nur die Eingangssignale jener EDIF-Instanz eingetragen werden, welcher dieser VHDL-Prozeß zugeordnet ist.
9. Verfahren nach einem der voranstehenden Ansprüche, bei dem die Schaltung mindestens eines programmierbaren Bausteines simuliert wird.

Hierzu 1 Seite(n) Zeichnungen

Nummer: DE 44 08 106 A1
Int. Cl.[5]: G 06 F 15/60
Offenlegungstag: 15. Dezember 1994

Figur 1



– Leerseite –

DOCKET NO: GR99P5005
SERIAL NO: 09/680,370
APPLICANT: Schneider

LERNER AND GREENBERG P.A.
P.O. BOX 2480
HOLLYWOOD, FLORIDA 33022
TEL. (954) 925-1100